```
-- BootLoaderUtilities.mesa
-- Last Modified by Sandman,  May 10, 1978  1:38 PM

DIRECTORY
  AltoFileDefs: FROM "altofiledefs",
  BcdDefs: FROM "bcddefs",
  ControlDefs: FROM "controldefs",
  DirectoryDefs: FROM "directorydefs",
  FrameDefs: FROM "framedefs",
  LoaderBcdUtilDefs: FROM "loaderbcdutildefs",
  LoaderUtilityDefs: FROM "loaderutilitydefs",
  SegmentDefs: FROM "segmentdefs",
  StringDefs: FROM "stringdefs",
  SystemDefs: FROM "systemdefs";

DEFINITIONS FROM BcdDefs, LoaderUtilityDefs;

BootLoaderUtilities: PROGRAM
  IMPORTS DirectoryDefs, LoaderBcdUtilDefs, SegmentDefs, StringDefs,
    SystemDefs
  EXPORTS LoaderUtilityDefs = PUBLIC

  BEGIN

  files: FileTable ← NIL;
  loadee: LoaderBcdUtilDefs.BcdBase;
  ssb: BcdDefs.NameString;
  ftb: CARDINAL;
  nfilestofind: CARDINAL ← 0;
  tableopen: BOOLEAN ← FALSE;

  SubStringDescriptor: TYPE = StringDefs.SubStringDescriptor;
  bcd: SubStringDescriptor = [base: "bcd", offset: 0, length: 3];

  AddFileName: PUBLIC PROCEDURE [file: BcdDefs.FTIndex] =
    BEGIN
    p: FileTable;
    i, offset, length: CARDINAL;
    FOR p ← files, p.link UNTIL p = NIL DO
      IF file = p.file THEN RETURN;
      ENDLOOP;
    p ← SystemDefs.AllocateHeapNode[SIZE[FileTableObject]];
    p↑ ← [file: file, filehandle: NIL, ext: FALSE, link: files];
    files ← p;
    IF file = BcdDefs.FTSelf THEN
      BEGIN
      p.filehandle ← SegmentDefs.VMtoFileSegment[loadee].file;
      RETURN
      END;
    IF file = BcdDefs.FTNull THEN BEGIN  p.filehandle ← NIL;  RETURN  END;
    offset ← (ftb+file).name;
    length ← ssb.size[(ftb+file).name];
    FOR i IN [offset..offset+length) DO
      IF ssb.string.text[i] = '. THEN
        BEGIN  p.ext ← TRUE;  EXIT  END;
      ENDLOOP;
    nfilestofind ← nfilestofind + 1;
    RETURN;
    END;

  FindFileName: PUBLIC PROCEDURE [name: StringDefs.SubString, ext: BOOLEAN]
   RETURNS [found: BOOLEAN, file: FileTable] =
    BEGIN OPEN StringDefs;
    filename: SubStringDescriptor ← [base: @ssb.string, offset:, length:];
    FOR file ← files, file.link UNTIL file = NIL DO
      filename.offset ← (ftb+file.file).name;
      filename.length ← ssb.size[(ftb+file.file).name];
      IF LastCharIsDot[@filename] THEN name.length ← name.length + 1;
      IF ext = file.ext AND EquivalentSubStrings[@filename, name] THEN
        RETURN[TRUE, file];
      ENDLOOP;
    RETURN[FALSE, NIL];
    END;

  LastCharIsDot: PUBLIC PROCEDURE [name: StringDefs.SubString] RETURNS [BOOLEAN] =
    BEGIN
```

```
    RETURN[name.base[name.offset+name.length-1] = '.];
    END;

  FileNotFound: PUBLIC SIGNAL [name: STRING] RETURNS [file: FileHandle] = CODE;

  LookupFileTable: PUBLIC PROCEDURE =
    BEGIN
    p: FileTable;
    ssd: StringDefs.SubStringDescriptor;
    name: STRING ← [40];
    IF nfilestofind # 0 THEN DirectoryDefs.EnumerateDirectory[CheckOne];
    FOR p ← files, p.link UNTIL p = NIL DO
      IF p.filehandle = NIL AND p.file # BcdDefs.FTNull THEN
        BEGIN
        ssd ← [base: @ssb.string, offset: (ftb+p.file).name,
          length: ssb.size[(ftb+p.file).name]];
        name.length ← 0;
        StringDefs.AppendSubString[name, @ssd];
        p.filehandle ← SIGNAL FileNotFound[name];
        END;
      ENDLOOP;
    END;

  CheckOne: PROCEDURE [fp: POINTER TO AltoFileDefs.FP, name: STRING]
   RETURNS [found: BOOLEAN] =
    BEGIN OPEN StringDefs;
    i: CARDINAL;
    dirName: SubStringDescriptor;
    bcd: SubStringDescriptor ← [base: "bcd", offset: 0, length: 3];
    file: LoaderUtilityDefs.FileTable;
    FOR i IN [0..name.length) DO
      IF name[i] = '. THEN
        BEGIN
        IF name.length-i # 5 THEN GOTO UseWholeName;
        dirName ← [base: name, offset: i+1, length: 3];
        IF ~EquivalentSubStrings[@dirName, @bcd] THEN GOTO UseWholeName;
        dirName.offset ← 0;  dirName.length ← i;
        GOTO HasBCDExtension;
        END;
      REPEAT
        UseWholeName => NULL;
        HasBCDExtension =>
          BEGIN
          [found, file] ← FindFileName[@dirName, FALSE];
          IF found THEN RETURN [ThisIsTheOne[fp, file]];
          END;
      ENDLOOP;
    dirName ← [base: name, offset: 0, length: name.length-1]; -- ignore dot on end
    [found, file] ← FindFileName[@dirName, TRUE];
    RETURN [IF found THEN ThisIsTheOne[fp, file] ELSE FALSE];
    END;

  ThisIsTheOne: PROCEDURE [fp: POINTER TO AltoFileDefs.FP, file: FileTable]
   RETURNS [BOOLEAN] =
    BEGIN
    file.filehandle ← SegmentDefs.InsertFile[fp, SegmentDefs.Read];
    nfilestofind ← nfilestofind - 1;
    RETURN [nfilestofind=0];
    END;

  FileHandleFromTable: PUBLIC PROCEDURE [filename: BcdDefs.FTIndex]
   RETURNS [file: SegmentDefs.FileHandle] =
    BEGIN
    p: FileTable;
    FOR p ← files, p.link UNTIL p = NIL DO
      IF p.file = filename THEN
        RETURN[p.filehandle];
      ENDLOOP;
    RETURN[NIL];
    END;

  FinalizeUtilities: PUBLIC PROCEDURE =
    BEGIN
    p: FileTable;
    FOR files ← files, p UNTIL files = NIL DO
      p ← files.link;
```

```
      IF files.file # NIL AND files.filehandle.segcount = 0 THEN
        SegmentDefs.ReleaseFile[files.filehandle];
      SystemDefs.FreeHeapNode[files];
      ENDLOOP;
    tableopen ← FALSE;
    END;

  InitializeUtilities: PUBLIC PROCEDURE [bcd: LoaderBcdUtilDefs.BcdBase] =
    BEGIN OPEN SystemDefs;
    loadee ← bcd;
    ssb ← LOOPHOLE[loadee+loadee.ssOffset];
    ftb ← LOOPHOLE[loadee+loadee.ftOffset];
    IF tableopen THEN FinalizeUtilities[];
    tableopen ← TRUE;
    END;

-- Utility Routines

  EnterCodeFileNames: PUBLIC PROCEDURE [loadee: BcdBase] =
    BEGIN
    SegSearch: PROCEDURE [sgh: SGHandle, sgi: SGIndex] RETURNS [BOOLEAN] =
      BEGIN
      IF sgh.class = code THEN
        AddFileName[sgh.file];
      RETURN[FALSE];
      END;
    [] ← LoaderBcdUtilDefs.EnumerateSegTable[loadee, SegSearch];
    RETURN;
    END;

  END....
```